Towards a Model@runtime Middleware for Cyber Physical Systems

Francisco Javier Acosta Padilla, Frédéric Weis, Johann Bourcier

Towards a Model@runtime Middleware for Cyber Physical Systems

Francisco Javier Acosta Padilla
INRIA - University of Rennes 1
Campus de beaulieu
F35042 Rennes Cedex
francisco.acosta@inria.fr

Frederic Weis
IRISA - University of Rennes 1
Campus de beaulieu
F35042 Rennes Cedex
frederic.weis@irisa.fr

Johann Bourcier
IRISA - University of Rennes 1
Campus de beaulieu
F35042 Rennes Cedex
johann.bourcier@irisa.fr

ABSTRACT

Cyber Physical Systems (CPS) or Internet of Things systems are typically formed by a myriad of many small interconnected devices. This underlying hardware infrastructure raises new challenges in the way we administrate the software layer of these systems. Indeed, the limited computing power and battery life of each node combined with the very distributed nature of these systems, greatly adds complexity to distributed software layer management. In this paper we propose a new middleware dedicated to CPS to enable the management of software deployment and the dynamic reconfiguration of these systems. Our middleware is inspired from the Component Based Systems and the model@runtime paradigm which has been adapted to the context of Cyber Physical Systems. We have conducted an initial evaluation on a typical Cyber Physical Systems hardware infrastructure which demonstrates the feasibility of providing a model@runtime middleware for these systems.

1. INTRODUCTION

We are now surrounded by a plethora of interconnected devices (mobile phones, household appliances, wearable sensors, and so on) that continuously collect data on our living environment. This distributed computing infrastructure is becoming pervasive and users tend to naturally interact with it. This new computing environment offers a lot of opportunities for developing new applications in many different domains. Technology can now operate behind the scene by dynamically responding to people wishes. Based on this principle, Cyber Physical Systems (CPS) have emerged. CPS are pervasive and long living systems formed by a constellation of many small interconnected devices integrated into houses, building, cities, factory chain, etc.

In a building automation scenario, CPS typically rely on sensor nodes that detect and record data such as presence, temperature, ambient lighting and energy consumption. Thus, a CPS uses sensors to continuously analyse the situation in order to adapt our living environment to match user needs and preferences. User wishes may involve different objectives such as comfort, air quality, and energy savings. To go beyond energy management and comfort, building automation systems have to deal with new types of services, depending on the use of the building: fire safety and security management for hotel, indoor air quality control in schools and office buildings, etc. The opportunities of services offered by CPS and the user preferences are countless and will change over the lifetime of these systems. The set of small interconnected devices integrated into building can be seen as a computing infrastructure that can host these new services. Consequently the software deployed on these nodes needs to be dynamically reconfigured and re-deployed to meet the evolution of services and user preferences.

For this reason, the capacity of dynamically deploying and reconfiguring the software layer of CPS is a crucial feature. In this paper, we address the problem of enabling the deployment of a distributed software layer and its dynamic reconfiguration over a network of nodes featuring very limited resources: memory, processing power, bandwidth communication and energy autonomy.

To solve this problem, various techniques have been proposed, such as system image replacement [13] and virtual machines [15]. These approaches have two main drawbacks. First they are not efficient in terms of energy. Second while they are suited to deploy static applications, these are not convenient solutions for CPS, because CPS operate in dynamic environments in which tasks performed by each node must be easily and frequently adapted.

In this paper we present our initial results towards the design of a middleware featuring deployment and reconfiguration facilities over a Cyber Physical System. Our middleware aims at implementing the paradigm of model@runtime taking into account the stringent requirements of Cyber Physical Systems.

This paper is structured as follows. Section 2 presents the Kevoree component model. Section 3 details the challenges of mapping the model@runtime paradigm to microcontrollers, and explains how we implemented Kevoree on these very limited nodes. Our proposal is evaluated in section 4. Section 5 presents related work and Section 6 gives our conclusion and highlights some perspectives to be addressed in future work.

2. BACKGROUND

The problem of managing software layer in a distributed system is not a new topic. Component based systems and the model@runtime paradigm [20] has been proposed in the context of distributed systems to manage the software layer. In this section, we briefly present these paradigms and highlight their limitations regarding the Cyber Physical Systems environment.

2.1 Component-based software architecture

The software architecture aims at reducing complexity through abstraction and separation of concerns by providing a common understanding of component, connector and configuration [6, 18, 24]. One of the benefits is that it facilitates the management of dynamic applications, which becomes a primary concern in Cyber Physical Systems [21]. Such systems are inherently dynamic and thus require techniques for self-adaptation. Many works [9, 11] have demonstrated the benefits of using component-based approaches in such open-world environments [3].

To satisfy the needs for adaptation, several component models provide solutions to dynamically reconfigure software architectures through, for example, the deployment of new modules, the instantiation of new services, and the creation of new bindings between components. In practice, component-based (and/or service-based) platforms like Fractal [4], OpenCOM [5], OSGi [23] or SCA [22] provide platform mechanisms to support dynamic architectures.

2.2 Model@runtime

Built on top of dynamic component frameworks, the model@runtime paradigm refers to a model-driven approach that aims at reducing the complexity of dynamic adaptation in distributed systems. In practice, component-based platforms offer reflection APIs that allow introspection within the application (e.g., which components and bindings are currently in place in the system) and dynamic adaptation (e.g., changing the current components and bindings). While some of these platforms offer roll-back mechanisms to recover after an erroneous adaptation [17], the purpose of model@runtime is to prevent the system from actually enacting an erroneous adaptation. In other words, the "model at runtime" is a reflection model that can be decoupled from the application (for reasoning, validation, and simulation purposes) and then automatically resynchronized. The principle of model@runtime is illustrated on figure 1.

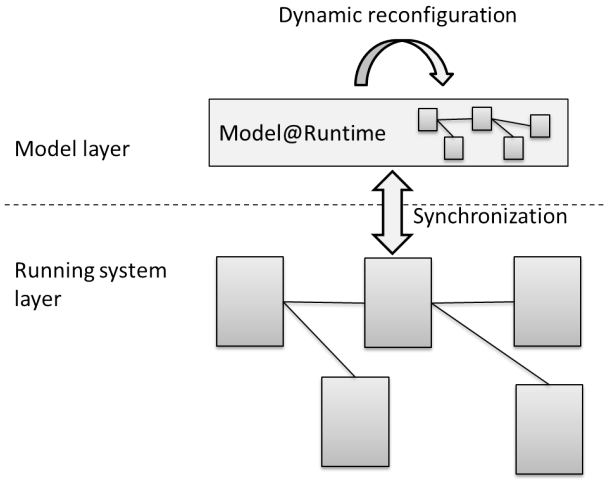

Figure 1: Model@runtime principle

2.3 The Kevoree framework

Kevoree is an open-source framework which provides an implementation of the model@runtime paradigm for distributed systems. It provides the following concept to design a distributed system featuring dynamic adpatation. The *Node* concept is used to model the infrastructure topology and the *Group* concept is used to model the semantics of inter-node communication, particularly when synchronizing the reflection model among nodes. Kevoree includes a *Channel* concept to allow for different communication semantics between remote *Components* deployed on heterogeneous nodes. All Kevoree concepts (*Component*, *Channel*, *Node*, *Group*) obey the object type design pattern [14] in order to separate deployment artifacts from running artifacts.

Kevoree supports multiple execution platforms (e.g., Java, Android, MiniCloud, FreeBSD, Arduino). For each target platform it provides a specific runtime container.

As a result, Kevoree provides an environment to facilitate the implementation of dynamically reconfigurable applications in the context of distributed system. Because our goal is to design and implement a middleware dedicated to CPS to enable the deployment and dynamic adaptations, the introspection and the dynamic reconfiguration facilities offered by Kevoree seems to suit our needs.

2.4 Limitations regarding CPS

The model@runtime paradigm has been mainly investigated in the context of distributed systems. These research efforts have been focused on the provision of a comprehensive set of tools to easily deploy, dynamically reconfigure, and disseminate software on a set of distributed computing units. The current model@runtime tools have been implemented regardless of the specific characteristics and constraints of a Cyber Physical System. In particular, the network topology and the resource constraint of the nodes forming the distributed system have not been taken into consideration. As a result, state of the art model@runtime tools are not suitable to be used in the context of Cyber Physical Systems. First, most approaches are relying on the Java language, which does not meet the resource constraint of the computing nodes. Secondly, the size of the model and its distribution among the system are not taking into consideration the limited memory capacity of each node, and their energy constraints.

In [10], an effort has been made to port the model@runtime paradigm on the constraints of a Cyber Physical System. Despite the particular attention given to the specific constraints of a Cyber Physical System, this work heavily relies on over the air firmware flashing to support the deployment and reconfiguration of software. We consider that relying on firmware flashing to support software deployment constitutes a flaw in the approach because of its energy cost (the complete firmware has to be sent, and if any error occurs, the whole process is restarted). A second limitation of this approach lies in the fact that each resource constrained node relies on a more powerful node to perform most of his task related to the dynamic reconfiguration (firmware synthesis, reconfiguration decision and so on). This second limitation is not suitable in the context of a system mainly composed of resource constrained nodes since all the resource constrained nodes have to be managed by bigger nodes. Pushing this idea further, the management of a CPS composed of a wide number of resource constrained devices and a bigger node, the latter will have to manage all the smaller devices in a centralized management scheme.

In this paper, we investigate a dedicated model@runtime approach for CPS which offers the same set of functionalities on resource constrained nodes and bigger nodes.

3. MODEL@RUNTIME FOR CPS

3.1 Overview

In this paper we present our initial result towards the design of a middleware which will offer the functionalities of model@runtime over a Cyber Physical System. Porting the concept of model@runtime on Cyber Physical Systems will enable a continuous management of the deployed software. This middleware will give a development framework for the applications development on top of these systems, and will provide the runtime infrastructure to support the deployment and dynamic reconfiguration of these applications.

As illustrated in figure 2, our goal is to provide a middleware that will be present on each node of the system, and will take care of the various tasks imposed by the model@runtime paradigm. More specifically, our middleware will be in charge of:

1. receiving new models that define the new targeted state of the system,
2. defining the set of local adaptations that are needed to reach this new state,
3. enacting the various local adaptations produced by the previous step.

The set of local adaptations enables the dynamic reconfiguration and may download new pieces of codes, instantiate or remove components and channels used to bind them, or reconfigure the value of any attribute.

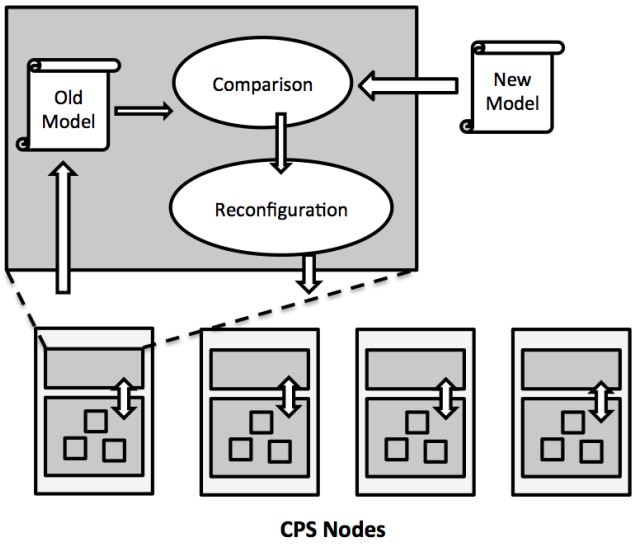

Figure 2: An overview of model@runtime for Cyber Physical Systems

3.2 Challenges

Designing the presented middleware in the context of Cyber Physical System raises scientific challenges. All the following challenges are related to the limited resources of these nodes and the particular topology of the network that interconnect them.

Currently the concept of model@runtime has been applied on top of object oriented languages which offers the required features at the language level to implement the model@runtime layer. The implementation of this model@runtime layer requires several features (such as dynamic code loading, network stack, etc.) from the operating system that operates on each node.

The first challenge lies in the fact that typical Operating Systems used in Cyber Physical Systems and which offer these features do not support these object oriented languages and thus the design must fit a procedural language such as C.

The second challenge lies in the very limited resources of these nodes thus forcing the middleware to meet the memory, and CPU constraints. This imposes constraints on the way the model is represented, stored and processed on each node. This challenge is particularly important and hard to meet, since many software components are needed to enable the communication in these environments (radio/MAC/IP stack and so on).

Another challenge comes with the energy constraint of each node, together with the mesh topology of the network. This implies the necessity to optimize the way the model and the software are disseminated in the network.

3.3 Current implementation

We use the ContikiOS [1] in order to provide an efficient way to distribute components, since it allows dynamic loading of binary modules. Contiki includes an implementation of the IPv6 compressed stack, 6loWPAN [19], which let us assign directly an IPv6 address. This enables a ready to use IoT environment. We use erbium's [16] CoAP implementation in order to have a REST engine which enables services discovery through the network. It is also used as a main communication channel between nodes. The dynamic code loading mechanisms are based on a dynamic linker and loader that use the standard ELF object file format [7].

The current implementation[1] is done using the version 4 of the Kevoree meta-model. This meta-model has been mapped into C code as a Contiki application. We are able to add nodes, components, groups and channels, and bind them through the Kevoree editor.

4. EVALUATION

This section describes the experiments conducted to evaluate our framework described in Section 3.3. The goal of this evaluation is to assess the feasibility of using a model@runtime implementation on CPS. In these experiments, we focus on measuring the overhead induced by our implementation, in order to evaluate the compatibility of this overhead with the resource constraints.

The platform used to run our experiments is located in a testbed called IoT-Lab [2]. IoT-Lab provides a very large scale infrastructure suitable for testing small wireless sensor devices and heterogeneous communicating objects. The M3 open node is a platform that provides similar resources which can be found in most of CPS. This node is based on a STM32 (ARM Cortex M3) microcontroller, which embed 512KB of flash memory and 64KB of RAM. The used toolchain to compile the source code is GCC for ARM.

This evaluation focuses on answering the two following research questions:

[1]The source code of our framework is available at: https://github.com/kYc0o/kevoree-contiki

RQ1: Does the overhead induced by our models@runtime implementation fit the resources constraints?

RQ2: Is this resulting overhead small enough to allow scalability?

To answer these questions four experiments are performed using the following metrics:

- **Start-up delay:** time needed to load the current model. To measure this time, we evaluate the time in milliseconds until the application is ready to work.
- **Consumption overhead:** amount of energy in joules drawn by the node while running our firmware. This energy is measured using IoT-Lab tools.
- **Memory:** amount of memory used by our model representation and our midlleware. That memory is measured by comparing both flash and RAM between various firmwares implementing our middleware, and another one without this implementation.

4.1 Overheads

Experimental setup. We compare the performances of two firmwares.

- The first firmware consists in a simple *Blink/COAP* application, with the network stack and a CoAP server initialized. This application consists in a LED that starts blinking until the end of the experiment.
- The second firmware includes the same functionalities but using our middleware. For this purpose we add a model@runtime platform in order to get a model reflecting the current state of the *Blink/COAP* application. To do so, a basic model is built from the current system, representing the *Blink/COAP* application as a component instance.

All the generated models are compatible with the version 4 of the Kevoree editor[2].

	Memory used		Energy consumption	Startup delay
	ROM (in bytes)	RAM (in bytes)	Joules	Msec.
Blink + CoAP	79344	13244	9.6	0
Kevoree + Blink + CoAP	112724	15822	9.606	39.1

Table 1: Memory use for the *Blink/COAP* example

In this experiment the two firmwares are uploaded on an IoT-lab node and the applications are executed during one minute. To evaluate the overhead of our middleware the two firmwares are compared with respect to:

- memory consumption both in ROM and RAM,
- the energy consumption,
- and the startup delay.

The result of this experiment are shown in Table 1.

As it can be observed from the table, the usage of a model@runtime has a visible overhead on the memory both in ROM and RAM. This overhead is due to the code of our middleware for the ROM part and to the model loaded in memory for the RAM part. We consider that this memory overhead is reasonable compared to the benefit of enabling dynamic reconfigurations.

Our approach also impacts the startup time and causes a very small delay before the application is ready. This delay is measured using timestamps. It is due to the time used by the processor, in order to load a model@runtime from the current application. This delay is considered reasonable as it is very small and it only impacts the initial loading of the application and has no effect during the normal operation.

As shown on table 1, the overhead of our framework on the energy consumption is very low. This consumption has been measured using the data generated by the IoT-Lab platform, which includes voltage and power used by the node in an experiment. The energy consumption overhead, shown in Table 1, is obtained as a product of the power in watts used by the node while loading the model@runtime, and the time needed before the application is ready and the LED is blinking. The overhead on energy consumption is only due to the extra computing power needed in the startup phase to load the model in memory.

This overhead evaluation highlights the feasibility of implementing a complete model@runtime middleware on CPS. The memory overhead is reasonable and fits with the resource constraints of the CPS nodes. We consider that the critical overhead for such system is the energy consumption, and our results show that it is marginally impacted by our middleware.

4.2 Scalability

In this experiment we evaluate the scalability of our approach by focusing on the memory needed to represent a large model. To do so, we first measure the memory size without any model loaded in node's memory. The command *size* of the ARM compiler was used to obtain that measure, since this application does not need dynamic memory allocation.

Our goal is to evaluate the biggest size that the model can reach. We progressively increase the model size by augmenting the number of nodes until running out of memory. Three variants of this experiment are run at last:

- with one component per node,
- with two components per node,
- and with three components per node.

Figure 3 shows the memory usage for each model depending on its size. These results show that our current implementation enables to scale the model up to 60 nodes with one component per node, and up to 37 nodes with three components per node. These numbers are encouraging, some tens of nodes enable pervasive systems and CPS to run in various environments like house, building, car, factory chain etc.

[2]The generated models are available at: https://github.com/kYc0o/kevoree-contiki

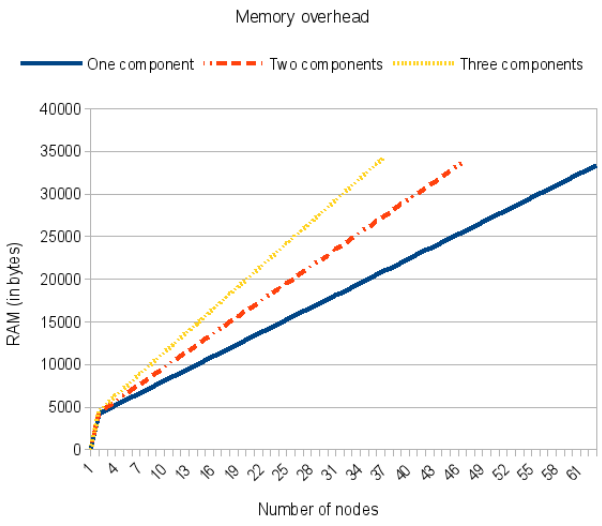

Figure 3: Memory overhead for 1, 2 and 3 *Blink/COAP* components.

4.3 Summary

Our initial results show that the models@runtime implementation is feasible for CPS and there are enough resources to deploy several other functional software components. It is possible to observe that our overheads are small enough to affect the overall operation of a node, while adding an abstract representation of the running system, in addition to reconfiguration and adaptation features. Since these results are promising, we showed that our middleware is able to manipulate a running application by affecting it through a component model, without an important overhead, neither in memory nor in energy consumption.

5. RELATED WORK

A preceding implementation of the models@runtime approach, called μ-Kevoree[10], is able to manage dynamic adaptations. When a change is made to the model, a new firmware with the changes is synthesized and sent over the network to the related nodes, while disseminating the reflected model to the entire network. This particular implementation provide the required functionality, but in some wireless networks where energy consumption plays an important role, this method becomes not adapted. Our goal is to provide a flexible and reconfigurable environment by sending to the nodes only the required software components, in order to provide new services. This will reduce the size of the transmitted binary, in such a way that network overhead is consequently reduced.

Another approach called Agilla, a mobile agent middleware designed to support self-adaptive applications in wireless sensor networks, proposes the distribution of specific applications in a WSN. Agilla's model focuses its design for localized adaptations like fire tracking. Thus, data collection is not one of its main features [8]. CPS in communicating environments demand an important amount of data collected from other devices. Due to this system requirements, distribution of single purpose agents seems not to be the best approach.

Valentine [12] is a component-based OS which aims to provide reconfiguration functions to WSN, using the Fractal [4] approach. It provides a component model that can be reconfigured using event-driven queues. The reconfigurations can only be made at specific predetermined points in time: when all tasks are finished. Since Valentine uses the Fractal approach components' introspection is available through the controller mechanism. These controllers can be used to obtain a complete image of the system, but increasing at the same time the processor use. Therefore, the time needed to build this representation will increase in relation with the architecture's complexity, which can be for instance the ubiquity of sub-components. This mechanism requires a more powerful node on which reconfiguration schemes can be settled. This important overhead does not lead us to have independent nodes in which a distributed system could be implemented. In most case studies, such distributed systems are often implemented, in order to deploy dynamic applications distributed among CPS. Since most of the current approaches aim to get highly reconfigurable and adaptable environments, it is possible to observe that CPS resource constraints are one of the most important issues. In our implementation, we are able to manage that issue, and we have made possible the usage of a model@runtime to reflect the system state and manipulate it as needed.

6. CONCLUSION

In this paper, we have presented our proposed solution to ease the deployment and enable the dynamic reconfiguration of the software layer on a Cyber Physical System. Our solution is based on a middleware which proposes an adoption of the model@runtime paradigm to the specific constraints of Cyber Physical Systems. We have presented the scientific challenges to implement this paradigm on resource constrained nodes and an initial evaluation of our implementation which demonstrates the feasibility of implementing this middleware on CPS.

This initial work opens many perspectives for CPS. Indeed, the possibility of reasoning on and managing the software layer presents on CPS after their initial deployment is a crucial enabler for many applications.

Our perspective includes works on the way to optimize the model storage, the software download and the propagation of changes within the system. All these aspects are impacted by the network topology and greatly influence the energy consumption of the system. Indeed, minimizing the communication will strongly impact the energy consumption of these systems.

7. REFERENCES

[1] Contiki: The open source os for the internet of things. `http://www.contiki-os.org`.

[2] Iot experimentation at a large scale. `https://www.iot-lab.info`.

[3] L. Baresi, E. Di Nitto, and C. Ghezzi. Toward open-world software: Issue and challenges. *Computer*, 39(10):36–43, Oct. 2006.

[4] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J. Stefani. The FRACTAL Component Model and its Support in Java. *Software Practice and Experience, Special Issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12):1257–1284, 2006.

[5] G. Coulson, G. Blair, P. Grace, A. Joolia, K. Lee, and J. Ueyama. A component model for building systems software. In *In Proc. IASTED Software Engineering and Applications (SEA'04*, 2004.

[6] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. An infrastructure for the rapid development of XML-based architecture description languages. In *24th International Conference on Software Engineering*, ICSE '02, pages 266–276, New York, NY, USA, 2002. ACM.

[7] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, Nov. 2006.

[8] C.-L. Fok, G.-C. Roman, and C. Lu. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Trans. Auton. Adapt. Syst.*, 4(3):16:1–16:26, July 2009.

[9] F. Fouquet, E. Daubert, N. Plouzeau, O. Barais, J. Bourcier, and J.-M. Jézéquel. Dissemination of reconfiguration policies on mesh networks. In *Distributed Applications and Interoperable Systems*, pages 16–30. Springer Berlin Heidelberg, 2012.

[10] F. Fouquet, B. Morin, F. Fleurey, O. Barais, N. Plouzeau, and J.-M. Jezequel. A dynamic component model for cyber physical systems. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, pages 135–144. ACM, 2012.

[11] V. Grassi, R. Mirandola, N. Medvidovic, and M. Larsson, editors. *Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering, CBSE 2012, part of Comparch '12 Federated Events on Component-Based Software Engineering and Software Architecture, Bertinoro, Italy, June 25-28, 2012*. ACM, 2012.

[12] N. Hoang, N. Belloir, C. Pham, and S. Sentilles. Valentine: A dynamic and adaptive operating system for wireless sensor networks. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2008, 28 July - 1 August 2008, Turku, Finland*, pages 1297–1302. IEEE Computer Society, 2008.

[13] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 81–94, New York, NY, USA, 2004. ACM Press.

[14] R. Johnson and B. Woolf. The Type Object Pattern, 1997.

[15] J. Koshy and R. Pandey. Vm*: Synthesizing scalable runtime environments for sensor networks. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, pages 243–254. ACM Press, 2005.

[16] M. Kovatsch, S. Duquennoy, and A. Dunkels. A low-power coap for contiki. In *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011)*, Valencia, Spain, Oct. 2011.

[17] M. Léger, T. Ledoux, and T. Coupaye. Reliable dynamic reconfigurations in a reflective component model. *Component-Based Software Engineering*, pages 74–92, 2010.

[18] N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Softw. Eng.*, 26:70–93, January 2000.

[19] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), Sept. 2007.

[20] B. Morin, O. Barais, G. Nain, and J.-M. Jezequel. Taming dynamically adaptive systems using models and aspects. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 122–132, Washington, DC, USA, 2009. IEEE Computer Society.

[21] E. D. Nitto, C. Ghezzi, A. Metzger, M. P. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 15(3-4):313–341, 2008.

[22] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani. A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures. *Software: Practice and Experience*, 2011.

[23] The OSGi Alliance. OSGi Service Platform Core Specification, Release 5.0, June 2012. http://www.osgi.org/Specifications/.

[24] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala Component Model for Consumer Electronics Software. *Computer*, 33(3):78–85, 2000.